VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

GRAPHICS INTRO 64KB USING OPENGL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MATEJ BOBUĽA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL MATÝŠEK**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20478/2017/xbobul02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Bobuľa Matej**
Obor: Informační technologie
Téma: **Grafické intro 64kB s použitím OpenGL**
Graphics Intro 64kB Using OpenGL
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte se s fenoménem grafického intra s omezenou velikostí.
2. Prostudujte knihovnu OpenGL a její nadstavby.
3. Popište vybrané techniky použitelné v grafickém intru s omezenou velikostí.
4. Implementujte grafické intro s použitím OpenGL, aby velikost spustitelné verze nepřesáhla 64kB.
5. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek pro prezentování projektu.

Literatura:

- dle pokynů vedoucího

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Matýšek Michal, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
L.S. 612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Táto bakalárska práca rieši problematiku tvorby grafického intra v OpenGL s obmedzenou veľkosťou na 64kB. Popisuje použité metódy, vďaka ktorým sme schopný takéto intro vytvoriť. Medzi tieto metódy patrí: procedurálne generovanie terénu, procedurálne generovanie textúr, osvetlenie, animácia a tvorba časticových systémov. Taktiež sa zaoberá metódami na redukciu veľkosti výslednej aplikácie ako nastavenie kompilátoru a komprimácia súborov. Výsledkom je nočná prechádzka po sopečnom ostrove. Vytvorené grafické intro svojou veľkosťou nepresahuje hranicu 64kB.

Abstract

This bachelor thesis deals with the problem of creating an OpenGL Graphical Intro with size limited to 64kB. It describes different techniques and methods used to create such intro. These methods are: procedural generation of terrain, procedural generation of textures, lighting, animation and particle systems. It also deals with size-reduction methods such as compiler configuration and file compression. The result of the intro is a stroll in the night on a volcanic island. Created graphics intro in its size does not exceed the given limit of 64kB.

Kľúčové slová

OpenGL, 64kB, grafické intro, Perlinov šum, procedrálne generovanie, Phongov osvetlovací model, GLSL, skybox, bibloarding, časticové systémy, delenie povrchov, Loopov algoritmus, kompresia

Keywords

OpenGL, 64kB, graphic intro, Perlin noise, procedural generation, Phong reflection model, GLSL, skybox, bibloadring, particle systems, subdivision surfaces, Loop algorithm, compression

Citácia

BOBUĽA, Matej. *Grafické intro 64kB s použitím OpenGL*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Matýšek

Grafické intro 64kB s použitím OpenGL

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Michala Matýška. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Matej Bobuľa
16. mája 2018

Poďakovanie

Srdečne by som sa v prvom rade chcel poďakovať Ing. Michalovi Matýškovi, ktorý mi v priebehu celého riešenia práce poskytoval odborné vedenie, užitočné rady a dojmy. Ďalej by som rád vyslovil vďaku aj svojej rodine a blízkym za psychickú podporu.

Obsah

Kapitola 1

Úvod

Bakalárska práca sa zaoberá problematikou ako vytvoriť grafické intro v OpenGL, ktoré bude mať obmedzenú veľkosť na 64 kB. Popisuje použité metódy, vďaka ktorým sme schopný takéto intro vytvoriť. Samotné zadanie práce bližšie neurčuje obsah tohoto grafického intra a čo konkrétne sa v ňom má zobrazovať. To dáva autorovi pomerne veľku voľnosť pri výbere jednotlivých metód, ktoré budú na zostrojenie takéhoto grafického intra použité.

Ako obsah scény tejto bakalárskej práce som sa rozhodol pre nočnú prechádzku sopečným ostrovom. V kapitole 2 sú popísané metódy potrebné pre zostrojenie takéhoto ostrova. Obsahuje metódy procedurálneho generovania terénu, osvetlenie takéhoto terénu pomocou Phongovho osvetľovacieho modelu, ďalej popisuje teóriu vytvárania skyboxu a časticových systémov s bilboardingom. Knižnici OpenGL je venovaná samostatná kapitola 3, v ktorej je popísaný jej vznik a aj čo nám táto knižnica ponúka. V kapitole 4 je popis knižníc využitých pri riešení práce a popis implementácie metód z predošlej kapitoly. Kapitola 5 popisuje princípy redukcie celkovej veľkosti aplikácie aby bol veľkostný požiadavok na 64 kB splnený. Je v nej bližšie rozvinuté nastavenie prekladača v ktorom bola aplikácia vyvíjaná ako aj metódy komprimácie výslednej aplikácie. S výsledkom bakalárskej práce a jej testovaním sa stretneme v kapitole 6. Posledná kapitola 7 zhrňuje použité metódy a nimi dosiahnuté výsledky, zároveň popisuje možnosti jej vylepšenia.

1.1 Grafické intro

Grafické intro alebo Grafické demo je spustiteľný program, ktorého hlavnou úlohou je vytvoriť čo najpútavejšiu scénu, s limitovanou veľkosťou na diskovom rozhraní. Takto vytvorený program by mal demonštrovať fantáziu autora, jeho programárske schopnosti a zároveň aj výkon stroja (počítača), na ktorom bol program vytvorený.

Počiatky grafických intier siahajú do 80. rokov minulého storočia. Za vytváranie takýchto intier, demoscén, boli zodpovedný hlavne nelegálny hackery, crackery, ktorý prelamovali ochranný softvér jednotlivých programov. Aby sa zviditeľnili, tak začali pridávať k takto prelomeným softvérom svoje podpisy v podobe demoscén. Tie mali za účelom, podobne ako aj grafické intro, poskytnúť audio-vizuálny zážitok, pre pozorovateľa, ktorý si stiahol takto upravený (cracknutý) softvér [19].

Kapitola 2

Techniky použité v intre

V nasledujúcej kapitole sa môžeme stretnúť s popisom jednotlivých techník a metód využitých pri tvorbe grafického intra s obmedzenou veľkosťou.

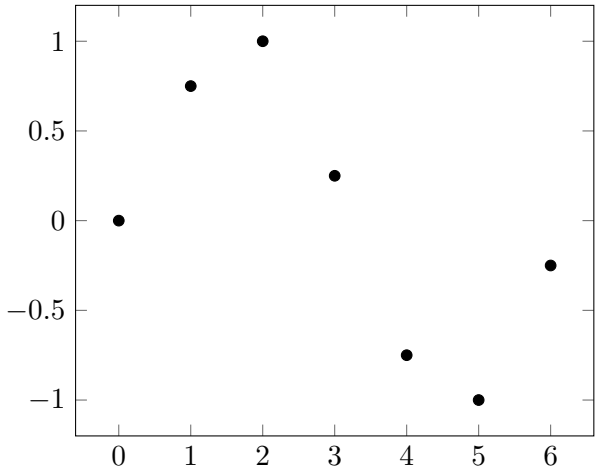
2.1 Procedurálne generovanie

Procedurálne generovanie je metóda ako vytvárať dáta algoritmicky a nie manuálne. V počítačovej grafike ho taktiež môžeme nazvať ako náhodné generovanie a bežne sa používa k vytváraniu textúr, prípadne 3D modelov. Vďaka procedurálnemu generovaniu sme schopný vytvoriť veľké množstvo obsahu (textúry, terén, ...), s malou veľkosťou v diskovom priestore. Keďže grafické intro má mať obmedzenú veľkosť, je nutné pri jeho tvorbe využiť práve metódy procedurálneho generovania. Hotové textúry a modely totiž môžu zaberať aj niekoľko megabajtov.

Pri procedurálnom generovaní sa využíva funkciu šumu a interpolačnú funkciu. Funkcia šumu v skutočnosti reprezentuje generátor náhodných čísel. Jeden z najstarších a najjednoduchších generátorov pseudonáhodných čísel je lineárny kongruetný generátor popísaný vzťahom:

$$X_{i+1} = (kX_i + c) \; mod \; m, \tag{2.1}$$

kde k, c, m su vhodne zvolené konštanty a X_0 je prvé číslo postupnosti. Generátor generuje celé čísla s rovnomerným rozložením v rozsahu $0 < x_i < m$. Dôležitou vlastnosťou funkcie použitej na generovanie náhodných čísel je determinizmus. Ak ju necháme prejsť dvakrát s rovnakými parametrami, dostaneme rovnaké výsledky.



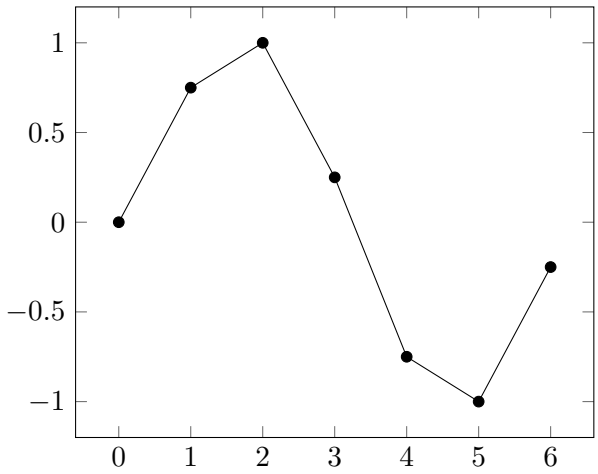
Obr. 2.1: Príklad generovania šiestich pseudoháhodných čísel v intervalu <-1,1>

Na takto vygenerované body potom aplikujeme vhodne zvolenú interpolačnú funkciu. Tá nám poslúži na prepojenie vygenerovaných bodov funkciou šumu. Existuje mnoho typov interpolácií, ktoré sa od seba výrazne odlišujú. Medzi najbežnejšie interpolačné metódy patrí lineárna a kosínusová interpolácia.

2.1.1 Lineárna interpolácia

Lineárna interpolácia je tá jednoduchšia metóda s predom spomenutých metód. Lineárnou interpoláciu medzi dvoma bodmi x, y je priamka. Lineárna interpolácia je z hľadiska matematických výpočtov pomerne rýchla a jednotlivé body na vzniknutej priamke sa dajú vypočítať pomocou vzorca 2.2. Nevýhodou tejto interpolácie sú ostré hrany ako vidieť aj na obrázku.

$$\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0} \tag{2.2}$$

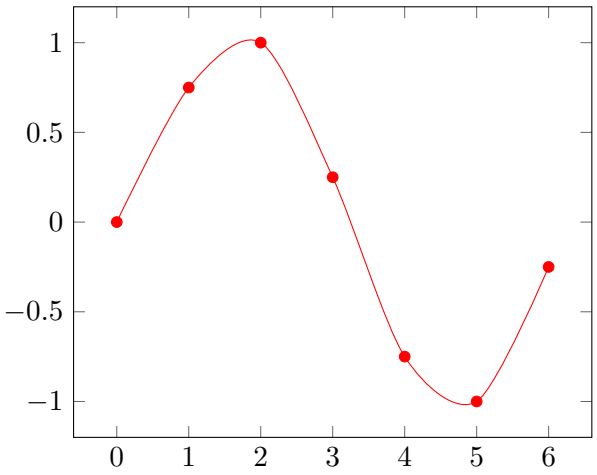


Obr. 2.2: Príklad lineárnej interpolácie medzi bodmi

2.1.2 Kosínusová interpolácia

Kosínusová interpolácia rieši odstránenie ostrých prechodov v lineárnej interpolácie na úkor rýchlosti výpočtu. Výsledkom, ako je znázornené na obrázku 2.3, sú omnoho jemnejšie prechody medzi jednotlivými bodmi. Na výpočet kosínusovej interpolácie použijeme nasledujúcu rovnicu:

$$f(x) = \frac{1 - cos(x\pi)}{2}. \tag{2.3}$$



Obr. 2.3: Príklad kosínusovej interpolácie medzi bodmi

2.1.3 Perlinov šum

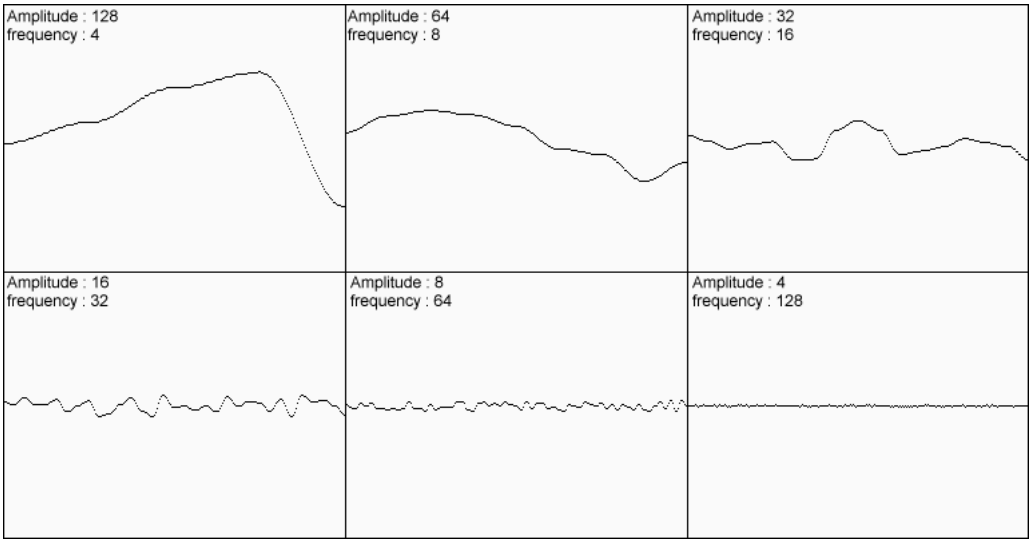
Perlinov šum je algoritmus, ktorý sa bežne používa v počítačových hrách alebo filmoch na procedurálne generovanie obsahu [2]. Je výsledkom práce Kena Perlina, ktorý ho vyvinul v roku 1985. Na jeho vytvorenie Perlinovho šumu potrebujeme vyššie spomenutý deterministický generátor náhodných čísel a kosínusovú interpoláciu. Princípom vytvárania Perlinovho šumu je vygenerovanie určitého počtu vrstiev (oktáv), ktoré su následne sčítané do jednej výslednej vrstvy. Každá vrstva závisí od predom určených premenných: amplitúda, frekvencia, perzistancia. Kde amplitúda je daná vzťahom:

$$a = perzistencia^i \tag{2.4}$$

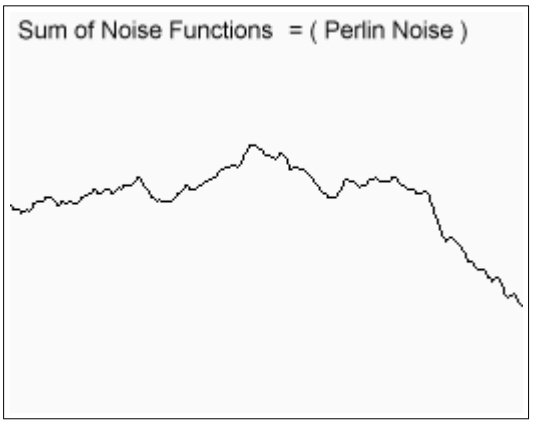
a frekvencia je vyjadrená vzťahom:

$$f = 2^i, \tag{2.5}$$

kde i je v oboch prípadoch oktáva. Počet iterácií algoritmu je daný počtom oktáv (vrstiev). Na nasledujúcich obrázkoch 2.6 a 2.5 môžeme vidieť generovanie niekoľkých vrstiev šumu a následné sšítanie do jednej vrstvy.



Obr. 2.4: Príklady šumu s rôznym nastavením frekvencie a amplitúdy, prevzaté z [2]

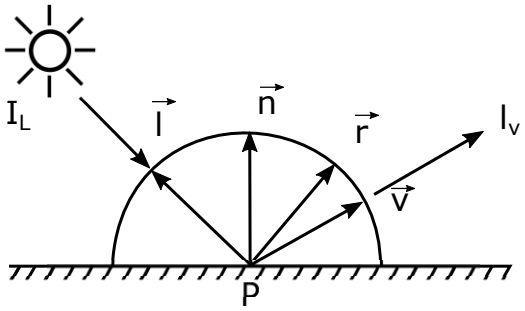


Obr. 2.5: Perlinov šumu získaný súčtom všetkých vrstiev z predošlého obrázka, prevzaté z [2]

2.2 Phongov osvetľovací model

Osvetlenie v skutočnom svete je veľmi komplikované a záleží na mnohých faktoroch. Kvôli jeho zložitosti, si ho ani v dnešnej dobe nemôžme v reálnom čase poriadne vykresliť kvôli limitácií hardvérom.

Z tohoto dôvodu je osvetlenie založené na aproximácií reality, využívajúcich zjednodušené modely. Sú oveľa jednoduchšie na spracovanie a pomerne vierohodne reflektujú svetlo v realite. Tieto modely sa nazývajú empirické modely. Jedným z týchto modelov je aj Phongov osvetľovací model [5]. Tento model navrhol vietnamský vedec Bui Tuong Phong v roku 1973.



Obr. 2.6: Vektory použité pri počítaní Phongovho osvetľovacieho modelu prevzaté z [27, strana 333]

Vektory s ktorými pracujeme pri počítaní jednotlivých zložiek sú na obrázku 2.6. Odraz na povrchu materiálu je určený smerom dopadajúceho svetla $\vec{l}$, smerom k pozorovateľovi l_v, bodom na povrchu P, normálovým vektorom v mieste dopadu $\vec{n}$ a zrkadlovo odrazeným paprskom $\vec{r}$ [27].

Základným stavebným kameňom tohoto modelu sú tri zložky: ambientná zložka, difúzna zložka, spekulárna zložka (anglicky: ambient, diffuse, specular). Vzťah na výpočet Phongovho osvetľovaciaho modelu potom vyzerá nasledovne:

$$I = L_A + L_D + L_S, \tag{2.6}$$

kde I je jednotka svietivosti, I_A značí ambientnú zložku, I_D značí difúznu žložku a I_S značí spekulárnu zložku.

2.2.1 Ambientná zložka

Jednou z vlastností svetla v reálnom svete je, že sa môže odrážať a lámať rozličnými spôsobmi. Toto spôsobuje, že sa môže dostať aj na miesta, ktoré nie sú priamo v ceste svetelného zdroja. Ambientná zložka Phongovho modelu reprezentuje túto vlastnosť svetla. V grafike predstavuje rovnomerné globálne osvetlenie objektu v scéne. Ambientnú zložku vypočítame vzťahom:

$$L_A = I_a * k_a, \tag{2.7}$$

kde I_a označuje intenzitu okolného osvetlenia scény a k_a je odrazivý koeficient materiálu objektu určujúci schopnosť povrchu odrážať okolné svetlo.

2.2.2 Difúzna zložka

Difúzna zložka predstavuje matné odlesky objektov v scéne. Plochy objektu, ktorých normálové vektory zvierajú zo svetelným lúčom čo najmenší uhol, budú najviac osvetlené. Výpočet difúznej zložky je vyjadrený vzťahom:

$$L_D = I_d * k_a * (\vec{n} * \vec{l}), \tag{2.8}$$

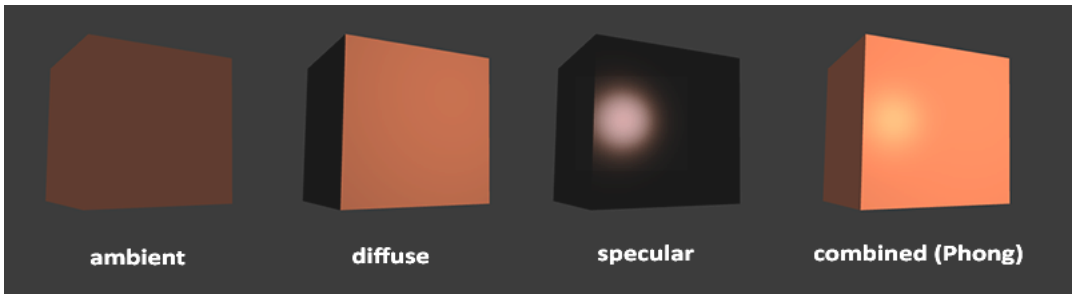
kde I_d je intenzita difúzneho svetla, hodnota k_a je rovnaká ako pri výpočte ambientnej zložky. Vektory $\vec{n}$ a $\vec{l}$ boli popísané na obr. 2.6.

2.2.3 Spekulárna zložka

Spekulárna zložka udáva intenzitu tej časti svetla, ktorá sa od objektu odráža prevažne v jednom smere podľa zákona odrazu. Spekulárna zložka určuje intenzitu odrazov svetla od zdroja smerom k pozorovateľovi. Je vyjadrená nasledujúcim vzťahom:

$$L_s = I_s * k_s * (\vec{v} * \vec{r})^n, \tag{2.9}$$

kde I_s označuje intenzitu odleskov, k_s je odrazivý koeficient určujúci mieru zastúpenia odrazenej lesklej zložky v celkovom odrazenom svetle. Vektory $\vec{n}$ a $\vec{r}$ boli popísané na obr. 2.6. Exponent n nám určuje ostrosť odrazu. Na obrázku 2.7 môžeme vidieť jednotlivé zložky Phongovho osvetľovacieho modelu.



Obr. 2.7: Zloženie Phongovho osvetlovacieho modelu prevzaté z [5]

2.3 Skybox

Skybox je metóda na vytváranie vzdialeného okolia scény v počítačovej grafike. Keď vytvárame skybox, tak v skutočnosti uzavierame scénu do kocky. Obloha, vzdialené hory a poprípade ďalšie objekty, s ktorými sa nebudeme nijako stýkať, sú nanesené na jednotlivé steny kocky. Týmto vytvárajú dojem vzdialeného trojrozmerného prostredia. Podobnú techniku popisuje aj skydome avšak namiesto kocky nanášame textúru na guľu alebo ovál. Pri mapovaní textúry na skybox využívame špeciálne textúry nazývane cube-maps. Tieto textúry sú rozdelené na plochy, kde každá plocha zodpovedá jednej stene kocky. Príklad takejto textúry môžeme vidieť obrázku 2.9.

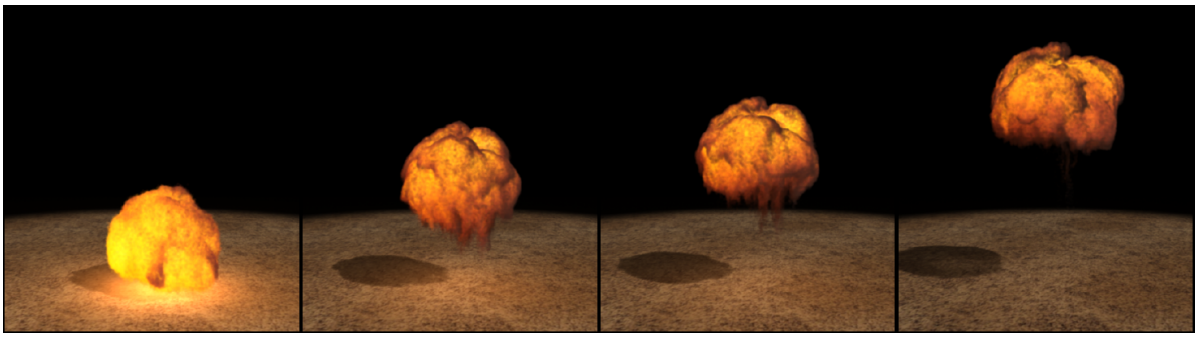
Obr. 2.8: Textúra skyboxu s popisom jednotlivých stien prevzaté z [14]

2.4 Časticové systémy

Časticové systémy je modelovacia a zobrazovacia technika, pomocou ktorej sme schopný reprezentovať príliš členité objekty alebo objekty, ktoré sa menia takým spôsobom že ich nie je možné reprezentovať ako povrch [27, strany 288 – 293]. Medzi takéto objekty patrí napríklad oheň, hmla, dážď, oblaky, explózie a pod. Podstatu časticového systému tvoria jednotlivé častice, s ktorých sa skladá.

Častice sú vlastne veľmi malé prvky príslušného časticového systému a každá jedna častica má určité vlastnosti definované týmto systémom. Medzi najdôležitejšie vlastnosti väčšinou patrí poloha, smer a rýchlosť pohybu. Ďalej môžeme medzi vlastnosti zaradiť aj zrýchlenie, farbu, tvar, priehľadnosť, textúru a pod.

Časticové systémy sa stali veľmi populárnymi a ich využitie zasahuje do mnohých kategórií nielen v oblasti IT. Od modelovania fotorealistických scén, tvorby filmových efektov a efektov do hier, simuláciu fyzikálnych javov, simuláciu ekosystémov. Časticové systémy sa veľmi často používajú spolu s Bilboardingom, popísaným v nasledujúcej kapitole 2.5.

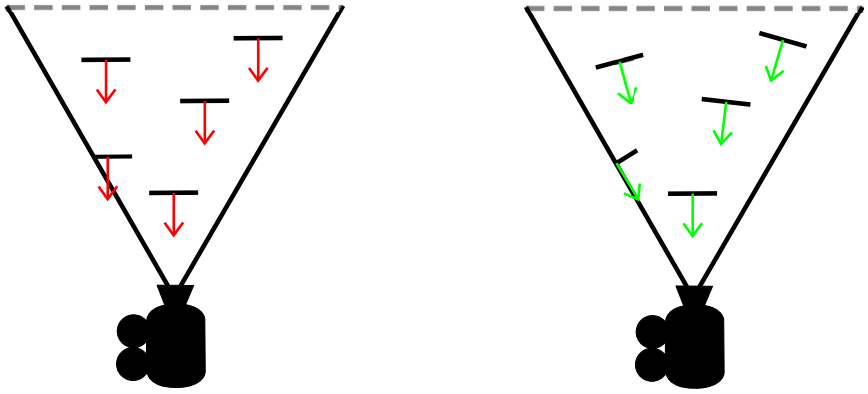


Obr. 2.9: Príklad časticového systému prevzaté z [8]

2.5 Bilboarding

Bilboarding je metóda, ktorá nastavuje orientáciu objektu tak, aby bol vždy otočený jedným smerom alebo otočený smerom k inému objektu. Obvykle sa však využíva, aby daný objekt smeroval čelom ku kamere. Technika je populárna v počítačových hrách a aplikáciách, ktoré vykresľujú veľké počty polygónov.

Týmto spôsobom sme schopný nahradiť zložitú geometriu v pozadí scény za jeden štvorec s nanesenou textúrou. Redukujeme tým počet polygónov v scéne niekoľkonásobne, keďže štvorec obvykle pozostáva iba z dvoch trojuholníkov. Na nasledujúcom obrázku 2.10 je znázornené natočenie objektov do kamery za pomoci bilboardingu. Ako už bolo spomínané v kapitole 2.4, bilboarding sa veľmi často využíva v kombinácií s časticovými systémami.



(a) Objekty v scnéne bez bilboardingu (b) Objekty v scnéne s bilboardingom

Obr. 2.10: Použitie bilboardingu

2.6 Instancing

Ďalšou užitočnou technikou pri tvorbe počítačovej grafiky je instancing [6]. Pomocou instancingu sme schopný vykresľovať väčšie počty objektov na rôznych miestach. Pri klasickom vykresľovaní väčšieho počtu objektov, by sme museli volať funkcie `glDrawArrays()` alebo `glDrawElements()` niekoľko krát za sebou, viď Algoritmus 1.

Algoritmus 1: Vykreslovanie pomocou glDrawArrays() (prevzaté a upravené z [6])

```
1  for ( i = 0; i < pocet_kresleni; i++)
2  {
3      nastav_novu_poziciu_objektu();
4      glDrawArray(GL_TRIANGLES, 0, pocet_vrcholov_objektu);
5  }
```

Ak by sme chceli vykresliť tisíce objektov a zachovať si slušný počet snímkov za sekundu (FPS), tak by sme týmto spôsobom pomerne rýchlo dosiahly limity nášho hardvéru. Dôvodom je, že pri použití týchto funkcií OpenGL musí cez procesor informovať grafickú kartu z ktorého buffer-u má čítať dáta a kde nájsť vlastnosti k jednotlivým vrcholom objektu.

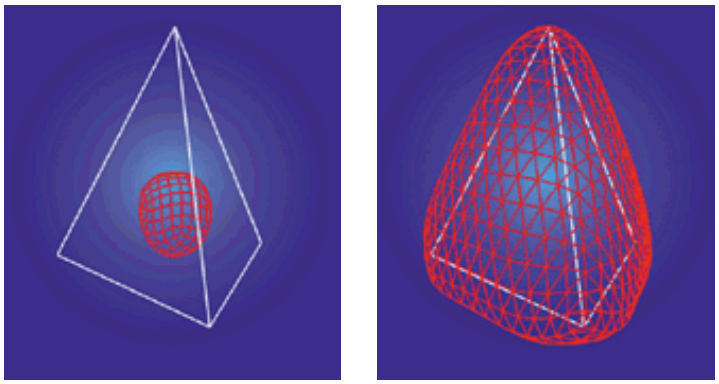
Vďaka instancingu sme schopný vykresliť niekoľkonásobne väčšie množstvo objektov naraz. Podstatou instancigu je totiž že predom pošleme na grafickú kartu tieto dáta a až

potom povieme OpenGL, že chceme vykresľovať objekt. Využívame pri tom funkciu `glDrawArraysInstanced` alebo `glDrawElementsInstanced`. Nastavujeme pri nich jeden špeciálny parameter, počet instacií, ktoré chceme vykresľovať. Takto sa pošlú všetky potrebné dáta na grafickú kartu iba jeden krát a zabraňuje sa tak relatívne pomalej komunikácií procesoru s grafickou kartou.

2.7 Delenie povrchov

Delenie povrchov (anglicky Subdivision surfaces), je v počítačovej grafike metóda vďaka ktorej sme schopný vyhladiť povrch objektu. Vstupom metódy je polygonálny mesh. Tento mesh sa potom rozdeľuje pomocou schém delenia. Schémy delenia nám určujú počet a pozíciu nových vrcholov, ktoré budú pridané do pôvodného meshu. Mnohonásobným aplikovaním tejto metódy na jeden mesh, sme schopný vyhladiť jeho povrch a zvýrazniť detaily. Nevýhodou delenia povrchov je však to, že pri každej iterácií nám vzniká nový počet vrcholov. Toto môže zapríčiniť spomalenie celého behu aplikácie v reálnom čase.

Schémy delenia môžeme vo všeobecnosti rozdeliť na: Aproximačné, Interpolačné. Rozdiel medzi týmito schémami je v tom, že u interpolačných schém vyžadujeme aby originálne vrcholy meshu ostali zanechané. U aproximačných schémach si prepočítavame aj pozície originálnych vrcholov. Porovnanie schém delenia môžeme vidieť na obrázku 2.12. V tejto bakalárskej práci sa používa Aproximačný Loopov algoritmus na vytváranie subdivízií, viď podkapitolu 2.7.1.



(a) Aproximačné schémy (b) Interpolačné schémy

Obr. 2.11: Porovnanie schém delenia prevzaté z [22]

2.7.1 Loop subdivision

Loopov algoritmus [20, strany 70 – 73] na vytváranie subdivizií je aplikovateľný iba na trojuholníkové meshe. Na začiatku, sa na každej hrane trojuholníka vytvorí nový bod. Súradnice nových bodov sú vypočítané podľa nasledovnej rovnice:

$$v = \frac{3}{8} * (a + b) + \frac{1}{8} * (c + d), \tag{2.10}$$

kde v je nový vrchol, a, b sú vrcholy hrany na ktorej vytvárame vrchol v. Vrcholy c, d sú zvyšné vrcholy trojuholníkov, ktoré majú jednu spoločnú hranu tvorenú vrcholmi a a b. Keďže Loop-ov algoritmus subdiviízi patrí do aproximačných schém musíme prepočítať aj

pozíciu starých vrcholov. Pozícia originálnych vrcholov je vyrátaná pomocou rovnice:

$$v = v * (1 - n * \beta), \tag{2.11}$$

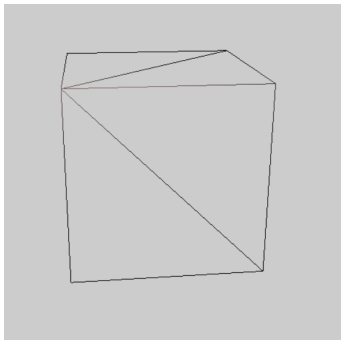
kde n nám určuje počet hrán pripojených k vrcholu. Hodnota β sa následne urči podľa hodnoty n. Loop originálne počítal hodnotu β takto:

$$\beta = \frac{1}{n}(\frac{5}{8} - (\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n})^2). \tag{2.12}$$

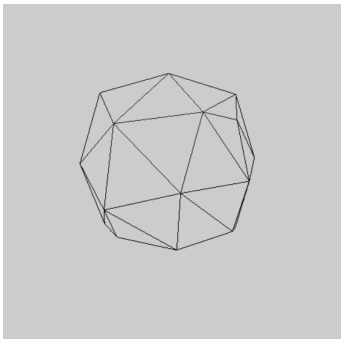
Ja som sa však rozhodol pre ľahšiu variantu kde sa hodnoty β určia nasledovne:

$$n = 3 \qquad \beta = \frac{3}{16}, \tag{2.13}$$

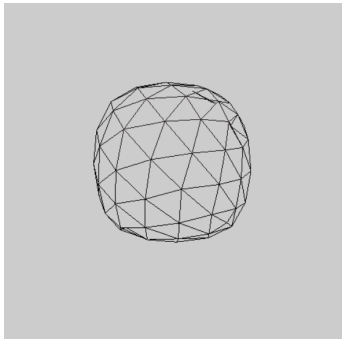
$$n > 3 \qquad \beta = \frac{3}{8n}. \tag{2.14}$$



(a) Základná kocka



(b) Prvá subdivízia



(c) Druhá subdivízia

Obr. 2.12: Vytváranie subdivízií pomocou Loopovho algoritmu

Kapitola 3

Knižnica OpenGL

OpenGL je skratka ktorá stojí za názvom Open Graphics Library. Jedná sa o platformne nezávislé rozhranie pre programovanie aplikácií (API), ktoré slúži na tvorbu 2D a 3D grafiky. Poskytuje nám stovky funkcií a procedúr, ktoré programu umožňujú pracovať s grafickou kartou počítača [21, strana 2].

S neustálym vylepšovaním počítačového hardware-u prišli aj grafické karty podporujúce 3D akceleráciu. S ich príchodom sa rozhodla firma Silicon Graphics Inc. (skratka SGI) vyvinúť nový otvorený štandard pre prácu s nimi. Firma v tej dobe už mala vyvinutý podobný štandard IRIS GL (Integrated Raster Imaging System Graphics Library) avšak vydať jeho otvorenú verziu nebolo možné z licenčných a patentových viazaností. Na základe predtým vytvoreného IRIS GL sa odvinula aj knižnica OpenGL. Hlavným obmedzením IRIS GL bolo že aplikácia mohla využívať iba programové rutiny podporované hardware-om. Toto sa v OpenGL podarilo eliminovať.

V súčasnosti existuje niekoľko verzií OpenGL. Od verzie OpenGL 2.0 máme k dispozícii aj jazyk OpenGL Shading Language (GLSL), ktorý si bližšie popíšeme v nasledujúcej podkapitole 3.1.

3.1 OpenGL Shading Language

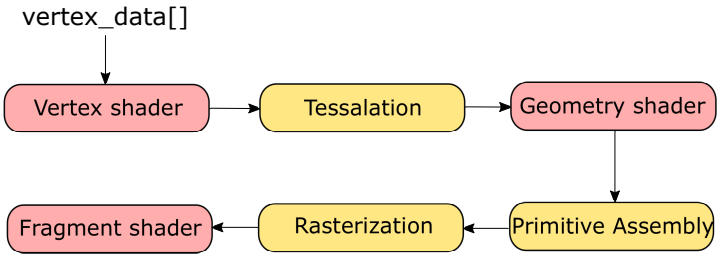
OpenGL Shading Language (ďalej len ako GLSL), je neoddeliteľnou súčasťou OpenGL. Jedná sa o programovací jazyk, pomocou ktorého vytvárame tzv. shadery viď kapitola 3.2. Tento programovací jazyk vychádza pôvodne z jazyka C a práve preto je mu aj veľmi podobný svojou syntaxou a sémantikou [21, strana 4]. Taktiež obsahuje skalárne dátové typy a operátory prebraté z jazyka C s výnimkou pointrov, ktoré v GLSL nie sú podporované.

Primárne bol tento jazyk navrhnutý na rýchle riešenie matematických operácií s vektormi a maticami. Môžeme sa stretnúť s vektormi, ktoré majú dva až štyri hodnoty `vec2()` - `vec4()` a maticami mierky 2x2 až 4x4 `mat2()` - `mat4()`. Ďalej obsahuje tento jazyk aj dátový typ `sampler`, slúžiaci na uloženie textúry.

3.2 Shadery

V počítačovej grafike je shader typ programu, ktorý bol pôvodne navrhnutý na tienenie (anglicky shading), čiže na nastavovanie príslušných hodnôt farby na snímke. Dnes shadery slúžia na radenie jednotlivých častí grafického reťazca (obr. 3.1) grafickej karty. Pre OpenGL sú napísané v jazyku GLSL.

Okrem už spomínaného radenia častí grafického reťazca shader-y vykonávajú rôzne druhy špecializovaných funkcií. Sú široko používané vo filmovej post-produkcií (anglicky postprocessing), počítačom generovaných snímok (anglicky computer-generated imagery, skratka CGI) a vo video hrách. Pri tvorbe tejto bakalárskej práci boli použité: vertex shader, geometry shader a fragment shader.



Obr. 3.1: Grafický reťazec

3.2.1 Vertex Shader

Vertex shader je program, ktorý sa zaoberá spracovaním individuálnych vrcholov vstupnej geometrie. Vo vertex shadery môžeme manipulovať s jednotlivými parametrami vertexu ako je napríklad farba, normála alebo pozícia. Medzi najčastejšie operácie patrí transformácia vrcholu do post-projekčnej scény. Vertex shader nám neumožňuje vrcholy pridávať ani odoberať.

3.2.2 Geometry Shader

Geometry shader, na rozdiel od vertex shadera, je voliteľným typom shaderu. Zaoberá sa spracovávaním primitív. Umožňuje pridávať alebo odoberať vrcholy primitíva, celé pirmitíva a tým ovplyvňovať výslednú geometriu. Geometry shader možno využiť napríklad na generovanie jednoduchej vegetácie, generovanie časticových systémov alebo na doplnenie detailov existujúceho modelu v reálnom čase.

3.2.3 Fragment Shader

Predošlé shadery sa zaoberali spracovaním a upravovaním geometrie. Fragment shader sa vykonáva až po rasterizačnej fázy grafického reťazca. Fragment shader nepracuje zo samotnou geometriou ale nad každým pixelom, ktorý naše primitívum po rasterizácií zaberá. Medzi najčastejšie operácie patrí aplikácie textúr, prelínanie a mixovanie textúr prípadne ďalšie modifikácie farby pixelu ako napríklad výpočet osvetlenia podľa osvetľovaciaho modelu.

Kapitola 4

Implementácia

Pri tvorbe grafického intra bol použitý programovací jazyk C++ a vývojové prostredie Microsoft Visual Studio 2015. Výsledné grafické intro využíva všetky metódy a techniky popísané v kapitole 3. Táto kapitola popisuje implementáciu a využitie týchto metód.

4.1 Použité knižnice

Knižnice sú neoddeliteľnou súčasťou pri vývoji akéhokoľvek softvéru v dnešnej dobe. Vytvorenie vlastných knižníc by bolo časovo veľmi náročné a preto je oveľa efektívnejšie použiť už existujúce knižnice. Nasledujúce podkapitoly popisujú knižnice, ktoré boli využité v tejto práci.

4.1.1 Windows API

Keďže samotné OpenGL nám nepodporuje vytváranie okna ale iba spôsob zobrazovania, je nutné okno vytvárať za pomoci inej knižnice. Jednou možnosťou by bolo využitie knižnice GLUT (OpenGL Utility Toolkit), ktoré by nám uľahčilo vytváranie potrebného okna, avšak by sa nám aj značne zvýšila celková veľkosť programu. Z tohoto dôvodu bolo využité Windows API, ktoré je súčasťou operačných systémov Microsoft Windows.

Windows API [17] nám umožňuje tvorbu a riadenie počítačových okien a sadu iných základných prvkov ako aj spracovanie vstupu z klávesnice a myši. Pomocou Windows API sa v tejto bakalárskej práci vytvárajú dve okná. Pri spustení sa najprv zobrazí malé okno s nastaveniami rozlíšenia [3] a následne na to okno, do ktorého sa bude vykresľovať grafický obsah [18].

4.1.2 OpenGL Mathematics

Ďalšou veľmi dôležitou knižnicou je OpenGL Mathematics [4], skratka GLM. Ako už z názvu vyplýva jedná sa o matematickú knižnicu. Bola navrhnutá v jazyku C++ a s úmyslom aby hocikto, kto pozná GLSL, bol schopný s ňou pracovať. Je to dosiahnuté tak, že jednotlivé funkcie a triedy majú rovnaké názvové konvencie ako v GLSL. Vďaka GLM sme schopný používať rovnaké dátové typy ako v GLSL napríklad vektory `glm::vec2()` - `glm::vec4()`, matice `glm::mat2()` - `glm::mat4()`. Zároveň nám umožňuje prevádzať nad týmito dátovými typmi sadu matematických funkcií a pri maticiach aj radu transformácií. V tejto bakalárskej práci sa knižnica GLM využíva hlavne k umiestňovaniu objektov do scény a to

aplikovaním rôznych transformácií na modelovú maticu objektu, nastavením perspektívnej projekčnej matice a na prepočítavanie pozície kamery v scéne.

4.1.3 OpenGL Extension Wrangler Library

OpenGL Extension Wrangler Library (skratka GLEW), je multiplatformná knižnica napísaná pre jazyky C/C++ [10]. Táto knižnica slúži ako rozšírenie ku OpenGL a stará sa o načítanie a overenie všetkých dostupných funkcií v OpenGL. Jej použitie je veľmi jednoduché, keďže jadro OpenGL aj spolu s rozšíreniami je uložené v jednom hlavičkovom súbore. Pre uľahčenie práce a zoznámenie sa s OpenGL, bola spočiatku pri riešení tejto bakalárskej práce používaná aj táto knižnica. Jej využitím však narastá výsledná veľkosť aplikácie. Z tohoto dôvodu bolo nutné vypustiť jej používanie a zabezpečiť načítanie potrebných funkcií manuálnym spôsobom. Na toto bola použitá funkcia `wglGetProcAddress( )`. Táto funkcia nám vracia adresy na rozširujúce funkcie OpenGL. Jej použitie je nasledovné:

```
glUseProgram = (PFNGLUSEPROGRAMPROC)wglGetProcAddress("glUseProgram");
```

Výpis 4.1: Príklad použitia funkcie `wglGetProcAddress( )`

Konštanty a funkcie, využité v tejto bakalárskej práci sú dostupné v hlavičkovom súbore `glext.h` [11].

4.1.4 Knižnica libv2

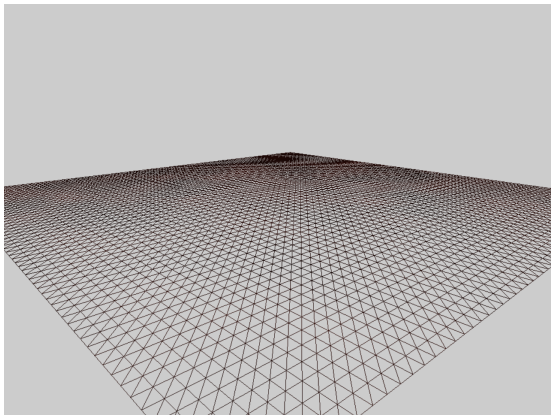
Aby nám výsledná aplikácia nepresiahla veľkosť 64 kB môžeme vylúčiť podobné formáty ako sú mp3, wav. Vhodným formátom z hľadiska veľkosti by bol formát MIDI. Tento formát v skutočnosti obsahuje iba zoznam nôt, ktoré sú prehrávané harvérom. To pre nás nie je zrovna najvhodnejšie, keďže na rôznych typoch hardvéru, to môže znieť rozdielne.

Knižnica libv2 je špeciálne zameraná na vytváranie a prehrávanie hudby pre grafické intrá s obmedzenou veľkosťou. Táto knižnica bola vyvinutá nemeckou skupinou programátorov pod menom Farbrausch [7]. Podporovaný formát knižnice je v2m. Ten nám poskytuje kvalitnejší zvuk ako MIDI a zároveň aj spĺňa veľkostné požiadavky.

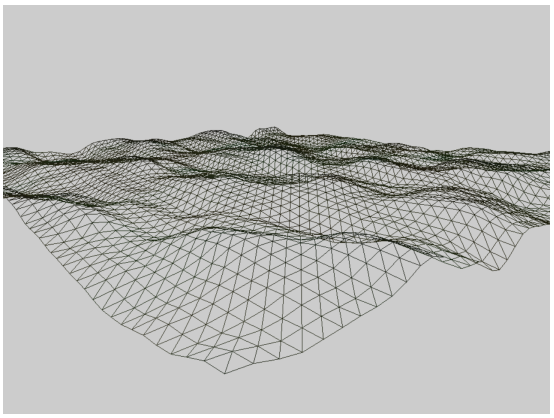
4.2 Generovanie terénu

Keďže mnou zvolená tematika tohoto grafického intra popisuje prechádzku po ostrove je nutné vygenerovať dostatočne veľký a dôveryhodný terén. Na jeho generovanie bol z tohoto dôvodu použitý Perlinov šum. V mnohých prípadoch generovania terénu sa najprv vygeneruje šum do textúry a následne sa táto textúra použije ako výšková mapa. Ja som sa rozhodol pre trošku iný prístup.

Terén má od začiatku stanovenú presnú veľkosť (premenná `TERRAIN_SIZE`) a presný počet vrcholov (premenná `VERTEX_COUNT`), z ktorých bude pozostávať. Podľa týchto údajov sa vygeneruje 3D mriežka, kde má každý vrchol nastavenú svoju Y-ovú súradnicu na 0, viď 4.1. Následne je na každý vrchol aplikovaná funkcia počítajúca výšku podľa Perlinovho šumu viď 4.2.

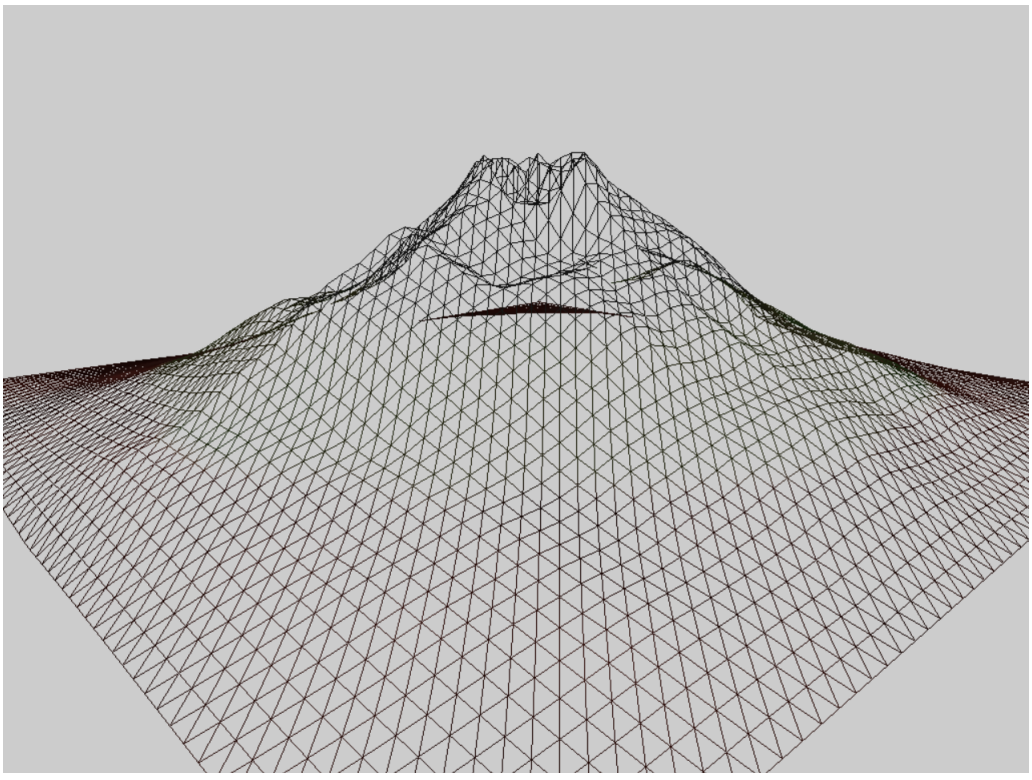


Obr. 4.1: Vygenerovaná mriežka



Obr. 4.2: Aplikácia šumu na mriežku

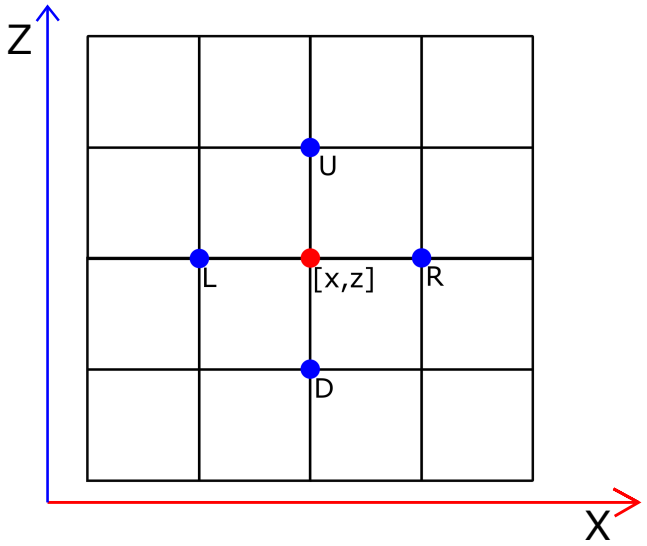
Týmto dostaneme pomerne realistický, kopcovitý terén. Aby sme dostali terén vo forme ostrova tak pre každý vrchol násobíme Y-ovú súradnicu hodnotou, ktorá je nepriamo úmerná zo vzdialenosťou od stredu terénu. Nakoniec sú vrcholy, v určitej nadmorskej výške, prevrátené tak, aby vytvárali priehlbinu. Zároveň si výšku jednotlivých vrcholov ukladám do dvojrozmerného poľa. Tá bude potom použitá pri počítaní kolízie s časticami, viď 4.8. Výsledný terén môžeme vidieť na nasledujúcom obrázku 4.3.



Obr. 4.3: Výsledný terén

Dôležitou súčasťou, ktorú musíme pri generovaní terénu taktiež počítať sú normálové vektory pre jednotlivé vrcholy. Tieto normálové vektory budú potom použité pri výpoč-

toch osvetlenia terénu. Výpočet normálového vektoru je závislí od jeho susedných vrcholov znázornených na obrázku 4.4.



$$L = vyska\_v\_bode[x-1, z]$$
$$R = vyska\_v\_bode[x+1, z]$$
$$D = vyska\_v\_bode[x, z-1]$$
$$U = vyska\_v\_bode[x, z+1]$$

Obr. 4.4: Vrcholy potrebné pre výpočet normály

Smer normály $\vec{n}$ vo vrchole zo súradnicami $[x, z]$ vypočítame potom nasledovne:

$$glm :: vec3 \quad Normal = glm :: vec3((L-R), 2.0f, (D-U)).$$

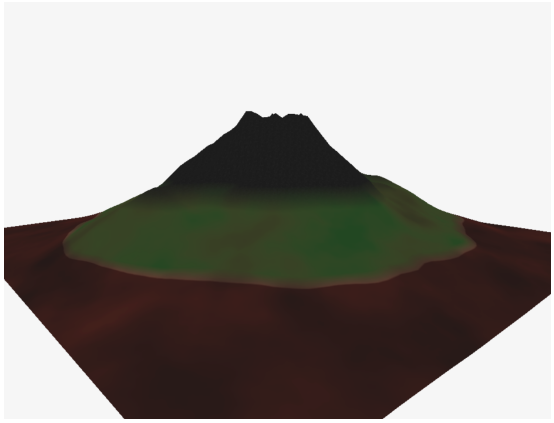
4.3 Osvetlenie

V scéne sa nachádzajú dva zdroje svetla. Smerové svetlo a bodové svetlo. Smerové svetlo využívame v prípadoch ak chceme navodiť dojem, že je svetelný zdroj vo veľmi veľkej vzdialenosti od plochy, ktorú osvetľuje. Toto sa využíva pri zdrojoch ako je slnko alebo mesiac. Nie je tomu inak ani v tejto bakalárskej práci. Smerové svetlo (directional light) osvetľuje všetky objekty v scéne rovnomerne. U bodových svetiel (point lights) je to trochu zložitejšie. Bodové svetlá šíria svoje lúče všetkými smermi do určitej vzdialenosti. U bodových svetiel musíme určiť pozíciu kde sa budú nachádzať a musíme prepočítavať aj vzdialenosť do akej jeho svetelné lúče budú siahať. Táto vzdialenosť F_{att} je vyrátaná podľa vzťahu:

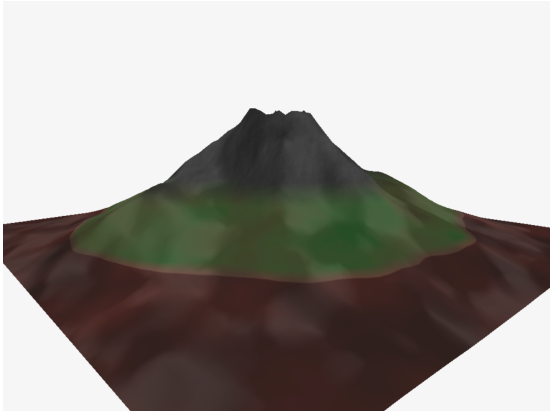
$$F_{att} = \frac{intesity}{K_c + K_l * d + K_q * d^2}. \tag{4.1}$$

Hodnota d reprezentuje vzdialenosť fragmentu od zdroja svetla. K_c je konštanta nastavená na hodnotu 1.0. Je to z toho dôvodu aby nám menovateľ nikdy neprešiel do záporných čisel. To by malo za následky zosilnenie svetla v určitých vzdialenostiach. K_l vynásobené vzdialenosťou d nám dáva lineárny pokles osvetlenia. K_q vynásobené d^2 nám reprezentuje kvadratický pokles osvetlenia. Hodnota *intensity* je premenná, ktorá sa mení v čase tak,

aby bodové svetlo vytváralo dojem blikania. Každá jedna zložka Phongovho osvetľovacieho modelu je potom vynásobená získanou hodnotou F_{att} a sčítaná dokopy. Všetky počty sa odohrávajú vo fragment shadery. Na obrázkoch 4.5 a 4.6 môžeme vidieť scénu bez osvetlenia a s osvetlením.



Obr. 4.5: Terén bez osvetlenia



Obr. 4.6: Osvetlený terén

Obr. 4.7: Bodové svetlo v scéne

4.4 Generovanie textúr

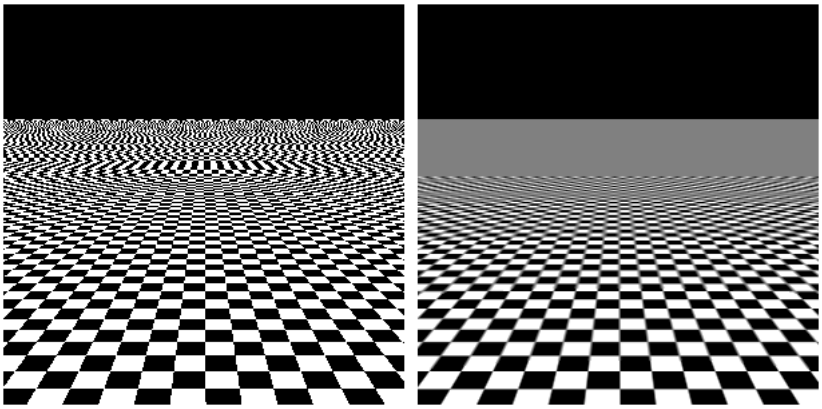
Aby sme jednotlivé objekty boli schopný v scéne od seba odlíšiť, môžeme jednotlivým objektom priradiť rôzne farby. Týmto by sme však vytvorili iba veľmi jednoduché a nereálne scény. Na detailnejší popis objektov v scéne nám slúžia textúry. Pomocou textúry sme schopný popísať nielen farbu objektu ale aj typ materiálu z ktorého sa skladá a aj mnoho ďalších vlastností. Vo väčšine prípadov, či už v počítačových hrách alebo filmoch, sa ako

textúry používajú statické obrázky a to veľmi často s vysokým rozlíšením. Tieto textúry môžu zaberať niekoľko kilobajtov až megabajty. V dnešnej dobe však nie je problémom si takéto textúry procedurálne vygenerovať a tým eliminovať nahrávanie objemných, externých súborov do aplikácie.

Textúry môžeme generovať pred začatím vykresľovania ako aj počas. Rozdiel je v tom, že textúry, ktoré sa nemusia nijako špeciálne meniť v čase si môžeme pripraviť ešte pred vykresľovaním. Generovať textúry počas vykresľovania totiž môže spomaliť celkový beh aplikácie. Príkladom takýchto textúr vygenerovaných v tejto bakalárskej práci sú: textúra trávy, omietka na dome alebo aj vlny na vode 4.5.

Príkladom textúr ktoré musíme generovať pri každom snímku vykresľovania je odraz na vodnej hladine viď 4.5. Odraz na vodnej hladine je závislý na pozícií kamery a tá v čase mení svoju polohu a smer pohľadu.

Generovanie textúr prebieha v dvoch fázach. V prvej fázy sa pripraví príslušný framebuffer objekt[1]. Tento framebuffer objekt má ku sebe pripnuté potrebné `GL_COLOR_ATTACHMENT0` na zapísanie do textúry a `GL_DEPTH_ATTACHMENT` na zapísanie hĺbkovej mapy. Do neho sa vykreslí celoplošný štvorec a výstup fragment shaderu je zapísaný do tohoto color attachmentu. V druhej fázy si získame takto vygenerovanú textúru z framebufferu pomocou `glReadPixels()` a vygenerujeme z nej mip-map textúru. Je to z toho dôvodu, že nie sme schopný vytvárať priamo mip-map textúry vo framebufferoch. Mip-mapping nám umožňuje používať menšie rozlíšenie textúr s narastajúcou vzdialenosťou viď 4.8.



Obr. 4.8: Na ľavo bez mip-mappingu. Na pravo s mip-mappingom. prevzaté z [24]

4.5 Generovanie vodnej hladiny

Keďže v scéne sa nachádza ostrov obklopený morom bolo potrebné generovať takú vodnú hladinu, ktorá by bola schopná čo najrealistickejšie kopírovať more skutočného sveta [9]. Takto vygenerovaná voda by mala byť schopná vlnenia, odrazu svetelných lúčov a odrazu objektov nad morskou hladinou. Vodná hladina sa skladá z množiny štvorcov tvoriacu jednu plochu. Na ne sú potom nanesené rôzne typy procedurálne generovaných textúr.

Ako prvé, sú pred začatím vykresľovania vygenerované dve textúry. Normálová mapa a uv-displacement mapa (ďalej už len ako DuDv mapa). Normálová mapa má v sebe ulo-

[1] užívateľom definovaný buffer, použity ako destinácia na vykresľovanie

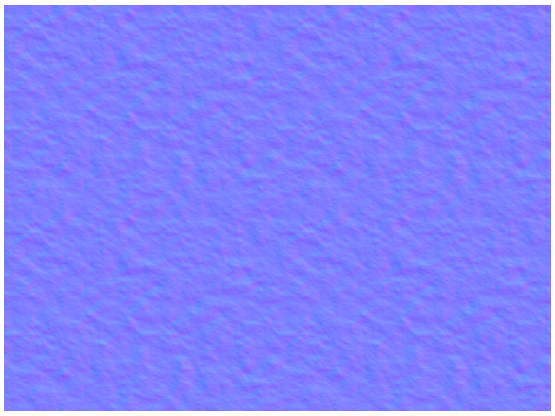
žené informácie o normálových vektoroch vo forme aditívneho farebného modelu RGB. Komponenty RGB reprezentujú (x, y, z) súradnice normálovych vektorov nasledovne [23]:

Os	**Hodnota**	**Farba veľkosť**	**Hodnota**
x	$< -1, 1 >$	Red	$< 0, 255 >$
y	$< -1, 1 >$	Green	$< 0, 255 >$
z	$< 0, -1 >$	Blue	$< 128, 255 >$

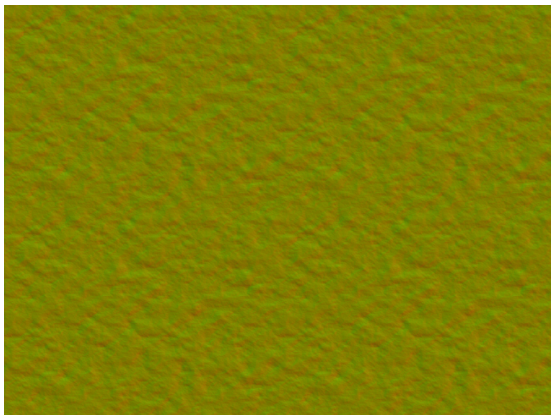
Tabuľka 4.1: Súradnice normálových vektorov mapované na komponenty RGB

Normála smerujúca od objektu má smer $(0, 0, -1)$ a je namapovaná na RGB hodnoty $(128, 128, 255)$. Z toho vyplýva že bežná farba normálovej mapy je svetlo-modrá. Vďaka využitiu normálovej mapy sa nemusíme zaoberať počítaním normál ako pri generácií terénu 4.2.

DuDv mapa je podobná normálovej mape v tom, že taktiež obsahuje informácie o vektoroch vo forme modelu RGB. Rozdielom však je, že vektory ktoré si získame z DuDv mapy sú použité na posunutie súradníc textúry v určitom bode. DuDv mapa v tomto grafickom intre je vytvorená invertovaním hodnôt normálovej mapy 4.9.



Obr. 4.9: Normálová mapa



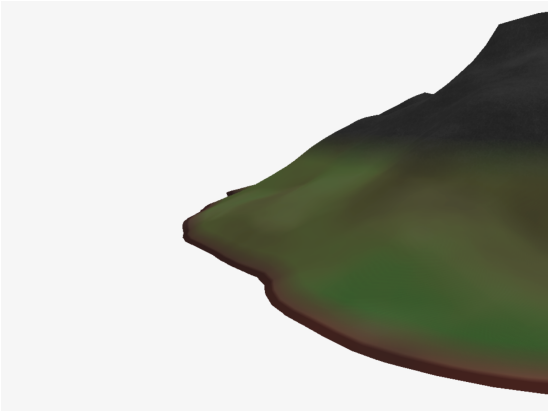
Obr. 4.10: DuDv mapa

Ďalšie dve textúry, ktoré sa nanášajú na vodnú hladinu, sa generujú pri každom snímku vykresľovania. Sú to textúry na odraz (reflekciu) a priehľadnosť (refrakciu) vody. Princípom generovania týchto textúr je, že si musíme vykresliť pre každú textúru celú scénu zvlášť. Pri generovaní reflekčnej textúry nás zaujíma iba to, čo sa nachádza nad vodnou hladinou. Pri generovaní refrakčnej textúry nás naopak zaujíma všetko pod vodnou hladinou. Aby sme nemuseli vykresľovať kompletne celú scénu dva krát, raz pre reflekciu a raz pre refrakciu, OpenGL nám umožňuje nastaviť si rovinu orezania. Pomocou nej sme schopný nepotrebné časti terénu orezať. Rovina orezania je daná rovnicou:

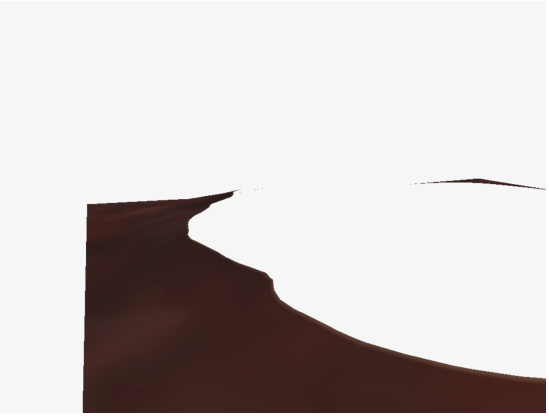
$$Ax + By + Cz + D = 0, \tag{4.2}$$

kde A, B, C sú normály roviny a D je výška v ktorej sa rovina nachádza. Hodnoty predáme vertex shaderu kde prebieha potom orezanie pomocou tejto roviny a premennej

`gl_ClipDistance[0]`. Na nasledujúcich obrázkoch 4.11, 4.12 môžeme vidieť orezaný terén.

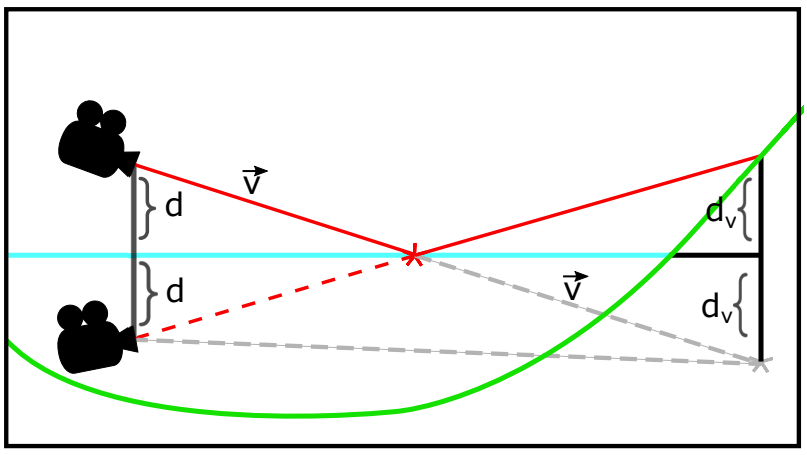


Obr. 4.11: Terén nad vodnou hladinou



Obr. 4.12: Terén pod vodnou hladinou

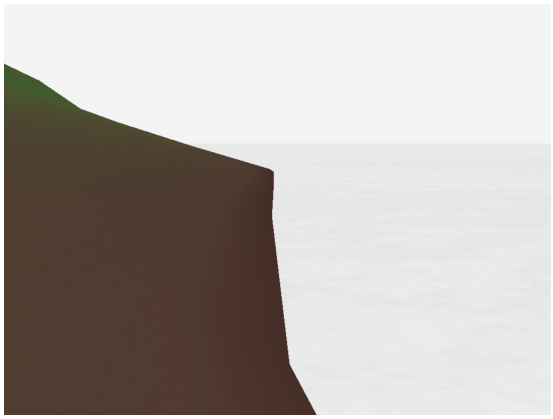
Ďalej je potreba zaistiť aby reflexná textúra správne reprezentovala odraz na vodnej hladine. Toto dosiahneme tým, že pred vykreslením scény do framebufferu si nastavíme kameru na novú pozíciu. Novú pozíciu kamery, sme schopný si odvodiť podľa nasledujúceho obrázka 4.13.



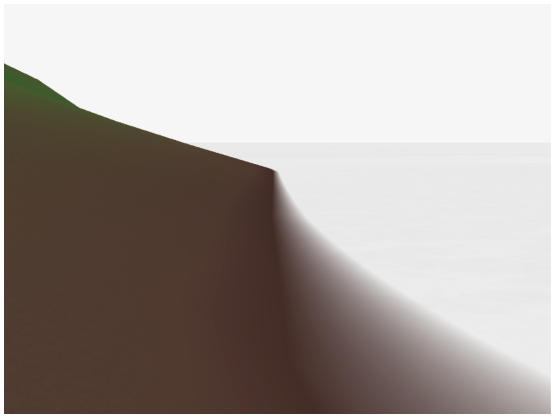
Obr. 4.13: Kamera pod hladinou vody

Najprv si získame najkratšiu vzdialenosť kamery od vodnej hladiny, hodnota d. X-ová a Z-ová súradnica sa nemení, Y-ová súradnica musí byť zmenšená o hodnotu $2 * d$. Vektor $\vec{v}$ nám určuje smer pohľadu z pôvodnej pozície kamery. Smer pohľadu sa opäť mení len v osy Y a vzdialenosť medzi vodnou hladinou a smerom pohľadu bude hodnota d_v. Pôvodný smer pohľadu zväčšíme len o hodnotu $2 * d_v$.

Pri generovaní refrakčnej textúry je taktiež generovaná aj hĺbková mapa. Hĺbková mapa je použitá na plynulý prechod medzi terénom a vodnou hladinou viď obrázky 4.14 a 4.15.

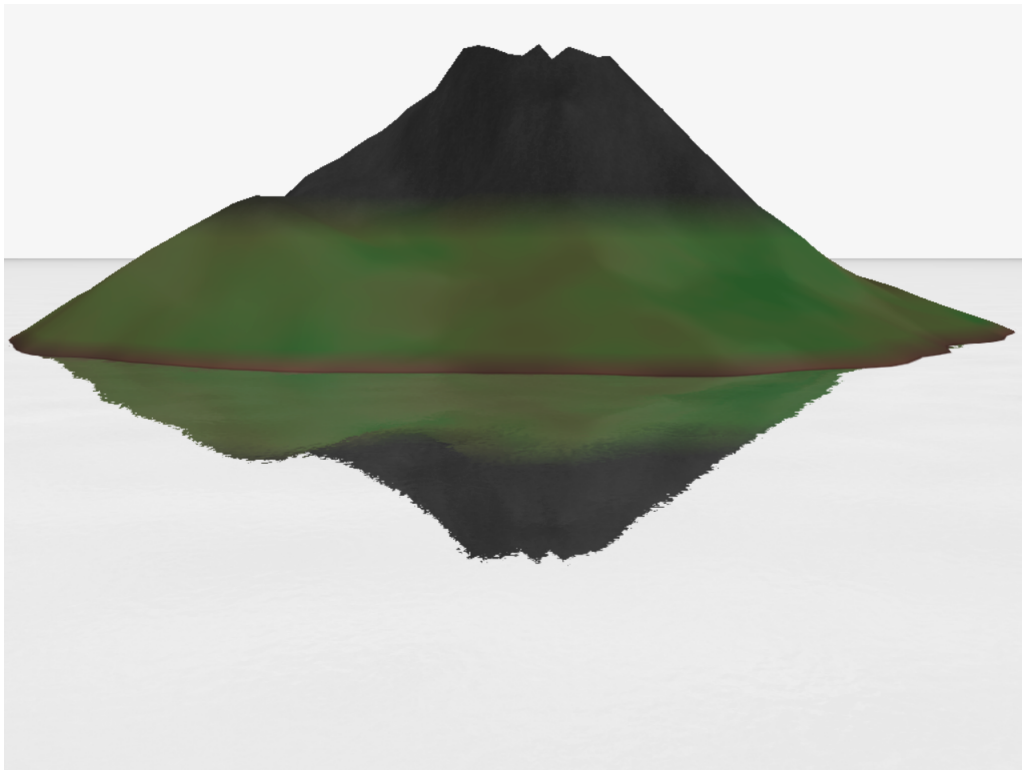


Obr. 4.14: Vodná hladina bez prechodu



Obr. 4.15: Vodná hladina bez prechodu

Výslednú vodnú hladinu docielime skombinovaním všetkých popísaných textúr vo fragment shadery, viď obrázok 4.16. Aby sme dosiahli efektu vlniacej sa vodnej hladiny je aplikovaný posun na súradnice DuDv textúry a normálovej textúry. Tento posun je predaný do fragment shaderu a inkrementuje sa s každou vykreslenou snímkou.



Obr. 4.16: Výsledná vodná hladina

4.6 Generovanie objektov v scéne

V scéne sa nachádzajú iba veľmi jednoduché objekty ako sú trojuholníky, kocky, kvádre a trojboký hranol. Jednotlivé vrcholy týchto objektov sú uložené v poli `GLfloat[]` štruktúrované nasledovne:

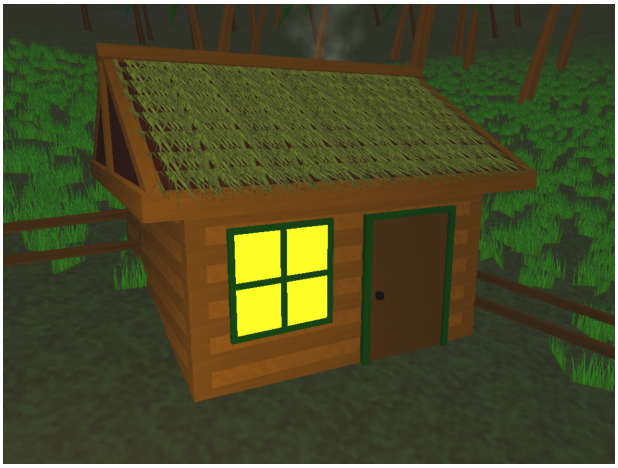
```
GLfloat nazov_objektu[ ]

{
        poz_x, poz_y, poz_z,    // pozícia vrcholu
        nor_x, nor_y, nor_z,    // normála vrcholu
        tex_x, tex_y          // súradnice pre textúry
}
```

Výpis 4.2: Príklad uloženia objektov v poli

Z tohoto poľa sú potom jednotlivé body načítané do VAO[2] aby mohli byť vykresľované. Zložitejšie modely v scéne sú zostrojené iba poskladaním tejto jednoduchej geometrie viď 4.6.1.

4.6.1 Domček

Model domčeku je zostrojený z jednej kocky, ktorá tvorí steny. Na túto kocku je nanesená textúra Perlinovho šumu. Strechu domu tvorí trojboký hranol. Rám okna, dvere, komín, drevené platne v stenách domu a na streche a aj plot sú všetky tvorené kvádrom s textúrou Perlinovho šumu. Transformačná matica použitá pri vykreslení kvádru je vytvorená pomocou funkcií knižnice `GLM`: `glm::translate()`, `glm::scale()`, `glm::rotate()`. Strecha je ďalej z prednej a zadnej strany vystlaná slamou aby to dodalo domčeku trošku tropickej atmosféry. Proces generovania slamy je rovnaký ako v kapitole 4.7.1. Vysledný domček možeme vidieť na obrázku 4.17.

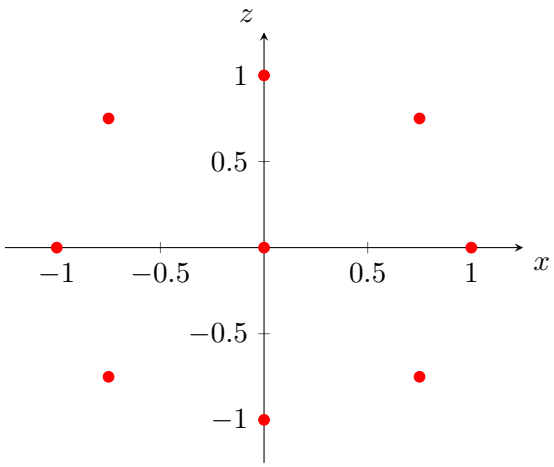


Obr. 4.17: Výsledný domček

[2] **V**ertex **A**rray **O**bject - objekt v OpenGL navrhnutý na uchovávanie informácií pre kompletné vykreslenie objektu

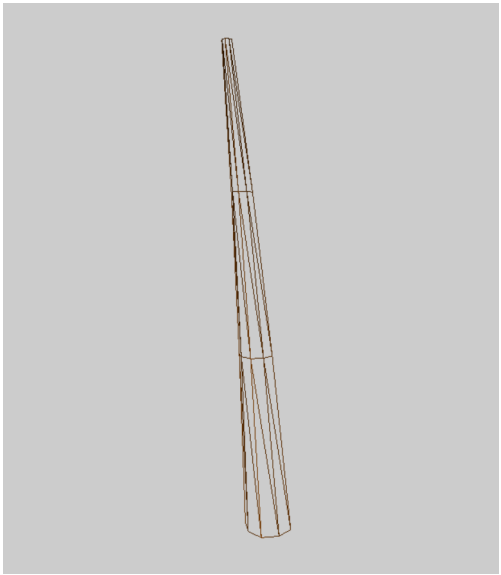
4.6.2 Palmy

Palmy v tejto bakalárskej práci sú tvorené z dvoch modelov. Model kmeňa stromu a model koruny palmy. Model kmeňa palmy, narozdieľ od ostatných modelov, je procedurálne generovaný. Najprv sa určí stupeň palmy. Stupeň nám určuje počet a výšku generovaných vrcholov. V každom stupni sa potom od počiatočného vrcholu vygeneruje osem ďalších podľa obrázka 4.18.



Obr. 4.18: Vygenerované vrcholy pre kmeň palmy s počiatočným bodom v strede súradnicovej sústavy

Pri každom stupni sa vzdialenosť nových vrcholov od stredu zmenšuje, a súradnice počiatočného bodu sú upravené tak, aby vygenerovaný kmeň nakoniec dával tvar ohnutej palmy, viď 4.19.



Obr. 4.19: Vygenerovaný kmeň stupňa 4

Model koruny palmy má vrcholy uložené v poli `GLfloat[ ]`. Pozícia, kam má byť koruna umiestnená, je získaná z modelovej matice kmeňa.

Pri vykresľovaní paliem je využité instancové vykresľovanie, kde sa najprv vygeneruje náhodné rozmiestnenie paliem a potom sa predá grafickej karte. Na nasledujúcom obrázku 4.20 je znázornená hotová palma.

Obr. 4.20: Hotová palma

Pre hotovú palmu je potom vo vertex shadery nastavený ustavičný pohyb pomocou funkcie `sin(time)` a `cos(time)`. Kde hodnota `time` sa inkrementuje pri každom snímku. Aby však nebol pohyb všetkých paliem rovnaký, intenzitou aj smerom, je táto hodnota `time` vynásobená náhodným číslom. Keďže GLSL nemá žiadnu funkciu na generovanie náhodných čísel, musíme si takú vytvoriť.

```
float rand(vec2 co){
       return fract(sin(dot(co.xy, vec2(12.9898, 78.233))) * 43758.5453);
}
```

Výpis 4.3: Funkcia na generovanie náhodných čísel v GLSL

4.7 Generovanie častíc

Model častíc v tomto grafickom intre je vytvorený pomocou geometry shaderu, popísaním v kapitole 3.2. Vstupom do neho je jeden vrchol. Z neho následne pričítaním vhodných X a Y súradníc vytvorím ďalšie štyri pomocou funkcie `EmitVertex( )`. Tieto štyri vrcholy sú pospájané do jedného štvorca. Štvorec je následne rasterizovaný a jeho fragmenty sú ďalej spracované vo fragment shadery.

4.7.1 Tráva

Pri generovaní trávy si pred vykresľovaním prepočítam pozíciu každého jedného štvorca. Tá je predaná grafickej karte. Pohyb trávy je zabezpečený v geometry shadery, podobne ako u

paliem, s využitím funkcie `rand( )`, viď 4.3, a fukcií `sin(time)` a `cos(time)`. Tento pohyb je aplikovaný iba na horné dva vrcholy modelu trávy. Potom je na náš štvorec nanesená textúra trávy, ktorá bola vytvorená ešte pred začatím vykresľovania vo fragment shadery. Podkladom pre textúru trávy sa stal program Ugly Grass od Erica Arnebäcka [1].

Obr. 4.21: Výsledná textúra trávy

Tráva má presne určený počet a pozíciu, ktorá sa v čase nemení. To nám narúša samotnú definíciu časticových systémov, avšak proces akým bola vytvorená im odpovedá.

4.7.2 Oheň

Narozdieľ od trávy, u ohňa musíme generovať častice pri každom snímku vykreľovania. Na začiatku každého vykresľovania sa vygeneruje určitý počet častíc ohňa. Jedna častica reprezentuje jeden plamienok a skladá sa z rovnakého modelu ako tráva. Každá častica má náhodne nastavený svoj smer, životnosť, silu gravitácie, veľkosť a rýchlosť akou zaniká. Pri vykresľovaní sa kontroluje veľkosť a životnosť častice. Ak častica dosiahne nulovú veľkosť alebo jej životnosť presiahne danú hodnotu je odstránená. Pred samotným vykreslením častíc je potrebné nastaviť `glEnable(GL_BLEND)` na miešanie farieb a vypnúť zapisovanie do depth bufferu funkciou `glDepthMask(FALSE)`. V práci je použité additívne miešanie farieb nastavené funkciou `glBlendFunc(GL_SRC_ALPHA, GL_ONE)`. Hotové ohnisko aj s osvetlením je na obrázku 4.7.

4.7.3 Láva

Častice lávy sú generované obdobným spôsobom ako častice ohňa. V scéne sa nachádzajú tri prúdy tečúcej lávy. Častice v jednotlivých prúdoch majú rovnaký smer. Narozdieľ od ohňa u častíc lávy dochádza ku detekcií kolízií. Pre každú časticu počítam aktuálnu výšku a ak náhodou je menšia alebo rovná výške terénu, tak ju nastavím tak, aby kopírovala výšku terénu. Zároveň aby sme nemali len tri prúdy, ktoré idú jedným určením smerom, je pri každom styku častice s terénom vypočítaná normála, ktorá mení smer pohybu toku. Normála je vypočítaná rovnako ako v kapitole 4.2, Y-ová súrdanica normály je potom prevrátená na zápornú aby smerovala dole do zeme.

4.8 Detekcia kolízií

Detekcia kolízií (anglicky collision detection), je matematický problém, pri ktorom sa rieši prienik dvoch objektov. V tejto bakalárskej práci riešim kolíziu časticových systémov s terénom.

Keďže náš terén je zložení zo samých trojuholníkov, ako prvú vec, čo musíme zistiť pri detekcií kolízií je, v ktorom trojuholníku dochádza ku kolízií z danou časticou. Do funkcie `collision_getHeight(float world_X, float world_Z)` sú predané súradnice častice, pre ktorú chceme vypočítať kolíziu. Z týchto súradníc, `world_X` a `world_Z`, si určíme konkrétnu mriežku terénu v ktorom sa častica nachádza. Jedna mriežka je tvorená dvomi trojuholníkmy terénu, viď obrázok 4.22.

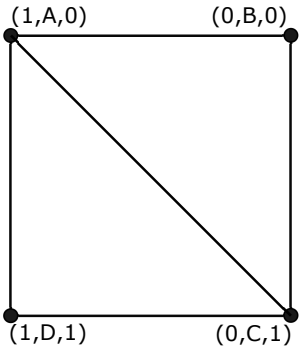
Aby nám častica neprepadla pod terén ale skutočne došlo ku kolízií stačí nastaviť výšku častice na výšku terénu vo vrchole [world_X, world_Z]. Pre výpočet presnej výšky v tomto vrchole použijeme prevod z Barycentrickej súradnicovej sústavy do Karteziánskej súradnicovej sústavy [13]. Prevod je realizovaný nasledujúcimi vzťahmi:

$$\lambda_1 = \frac{(v2_z - v3_z) * (pos_x - v3_x) + (v2_z - v3_z) * (pos_z - v3_z)}{(v2_z - v3_z) * (v1_x - v3_x) + (v3_x - v2_x) * (v1_z - v3_z)}, \tag{4.3}$$

$$\lambda_2 = \frac{(v2_z - v3_z) * (pos_x - v3_x) + (v2_z - v3_z) * (pos_z - v3_z)}{(v2_z - v3_z) * (v1_x - v3_x) + (v3_x - v2_x) * (v1_z - v3_z)}, \tag{4.4}$$

$$\lambda_3 = 1.0 - \lambda_1 - \lambda_2, \tag{4.5}$$

kde pos_x a pos_z sú súradnice vrcholu, ktorého výšku určujeme. Vrcholy $v1$, $v2$, $v3$ budú nadobúdať hodnoty podľa typu trojuholníku v ktorom sa častica nachádza podľa obrázka 4.22.



Obr. 4.22: Barycentrické súradnice na dvoch pravouhlých trojuholníkoch

Ak hľadáme bod ktorý leží v hornom pravouhlom trojuholníku z obrázka 4.22, budeme do vzorca dosadzovať hodnoty:

$$\begin{aligned} v1 &= (1, \quad vyska\_A, \quad 0), \\ v2 &= (0, \quad vyska\_B, \quad 0), \\ v3 &= (0, \quad vyska\_C, \quad 1). \end{aligned}$$

Pre spodný trojuholník by to boli hodnoty:

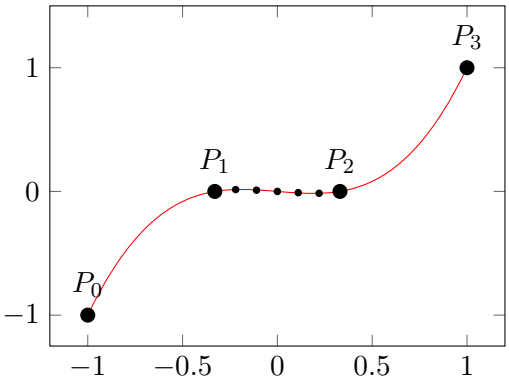
$$\begin{aligned} v1 &= (1, \quad vyska\_A, \quad 0), \\ v2 &= (1, \quad vyska\_D, \quad 1), \\ v3 &= (0, \quad vyska\_C, \quad 1). \end{aligned}$$

4.9 Kamera

Keďže grafické intro má byť dynamické bolo potrebné implementovať pohyblivú kameru. Spočiatku, pre účely ladenia a debugovania bola implementovaná kamera závislá na vstupe z klávesnice a myšky. Samotné intro však nemá byť interaktívne, preto bol implementovaný automatický pohyb kamery za pomoci Catmull-Rom krivky [25].

Catmull-Rom krivka je definovaná štyrmi kontrolnými bodmi P_0, P_1, P_2, P_3. Kde samotná krivka je kreslená len medzi bodmi P_1, P_2. Body P_0 a P_3 definujú zahnutie krivky. Funkcia na výpočet krivky si berie tieto body z predom definovaného pola `glm::vec3 cameraPositions[ ]` a generuje medzi nimi sadu nových bodov viď graf 4.23. Medzi nimi sa potom lineárne interpoluje. Počet vygenerovaných bodov je nastavený tak, aby boli prechody medzi nimi čo najplynulejšie a najjemnejšie.

Pohyb kamery v scéne som sa snažil prispôsobiť hudbe. Ak je hudba kľudná pohyb kamery je pomalší, naopak ak je hudba svižnejšia, zrýchli sa i pohyb kamery. Okrem pohybu kamery bolo potrebné taktiež nastavovať rôzne smery pohľadu. Ak sa kamera dostane do určitej pozície zmení sa aj smer pohľadu. Z tohoto hľadiska je intro rozdelené na štyri fázy. Kde v každej fázy je centrom pohľadu iný objekt.



Obr. 4.23: Príklad generovania bodov na krivke Catmull-Rom

4.10 Hudba

Účelom grafického intra je vyvolať v pozorovateľovi čo najlepší estetický dojem. Z tohoto dôvodu je správne zvolená hudba takmer kritickou súčasťou grafických intier. Pre prehrávanie hudby bola využitá vyššie spomenutá knižnice libv2 a formát v2m. K tejto knižnici je pridaný aj samotný prehrávač napísaný v jazyku C++. Vstupom do prehrávača je assemblerovký súbor, ktorý je nutný preložiť. Na jeho preloženie som si zvolil prekladač YASM [12]. Ako hudbu som si zvolil skladbu s názvom downdraft od skladateľa menom Teo [26].

Kapitola 5

Metódy na zníženie veľkosti aplikácie

Vzhľadom k tomu, že si to zadanie bakalárskej práce vyžaduje, je nutné klásť dôraz na výslednú veľkosť celého programu. Samotná veľkosť programu môže byť jednou z výhod, ktoré takto zamerané intrá ponúkajú. Čím sú programy menšie, tým sú rýchlejšie prenositeľné.

Na dosiahnutie čo najmenšej veľkosti je potrebné správne nastavenie prekladača. Ďalej je vhodné využívať datový typ `float`. Datový typ `float` je totiž najmenší dátový typ s plávajúcou desatinnou čiarkou. Pri jeho použíaní je nutné za číslom písať postfixovú značku **f**, napríklad 1.0f. Rozdieľ medzi datovým typom `float` a `double` sú štyri byty. Ak by výsledná veľkosť stále presahovala požadovanú, môžeme použiť tzv. exe-packery na jej redukciu.

5.1 Nastavenie prekladača

Ako už bolo spomenuté na začiatku kapitoly 4, na zhotovenia tejto bakalárskej práce sa použilo vývojové prostredie Microsoft Visual Studio 2015. Toto vývojové prostredie nám poskytuje množstvo nastaviteľných parametrov, ktoré nám ovplyvňujú tvorbu výslednej aplikácie či už, pri linkovaní súborov [16], generovaní kódu alebo kompilácií [15]. Správnym nastavením týchto parametrov sme schopný minimalizovať veľkosť výslednej aplikácie. Parametre, ktorých nastavenie ovplyvnilo výsledok tejto práce, sú nasledovné :

- /O1 - vytvára malý kód
- /Os - preferuje malý kód
- /fp:fast - nastavenie floating point modelu
- /GS- - vypnutie kontroly na buffery
- /INCREMETAL:NO - vypnutie inkrementálneho linkovania
- /DEBUG - vypnutie informácie pre debugger

5.2 Komprimácia

Keďže výsledná aplikácia po preložení presahovala veľkosť 64 kB, bolo nutné využiť aj komprimačné metódy. Medzi ne spadá využitie komprimačných programov, tzv. exe-packerov.

Exe-packery fungujú na princípe kombinovaia komprimovaných dát s kódom na dekompresiu. Po zabalení aplikácie sa na začiatok spustiteľného súboru, pridá kód na jej rozbalenie.

V nasledujúcej tabuľke 5.1 môžeme vidieť dva typy kompresných programov, ktoré boli testované v tejto bakalárskej práci.

Nástroj	**Parametre**	**Výsledná veľkosť**
Pôvodná veľkosť	–	276 kB
UPX 3.94w	-9	67 kB
UPX 3.94w	-ultra-brute	58 kB
MPRESS v2.19	-s	57 kB

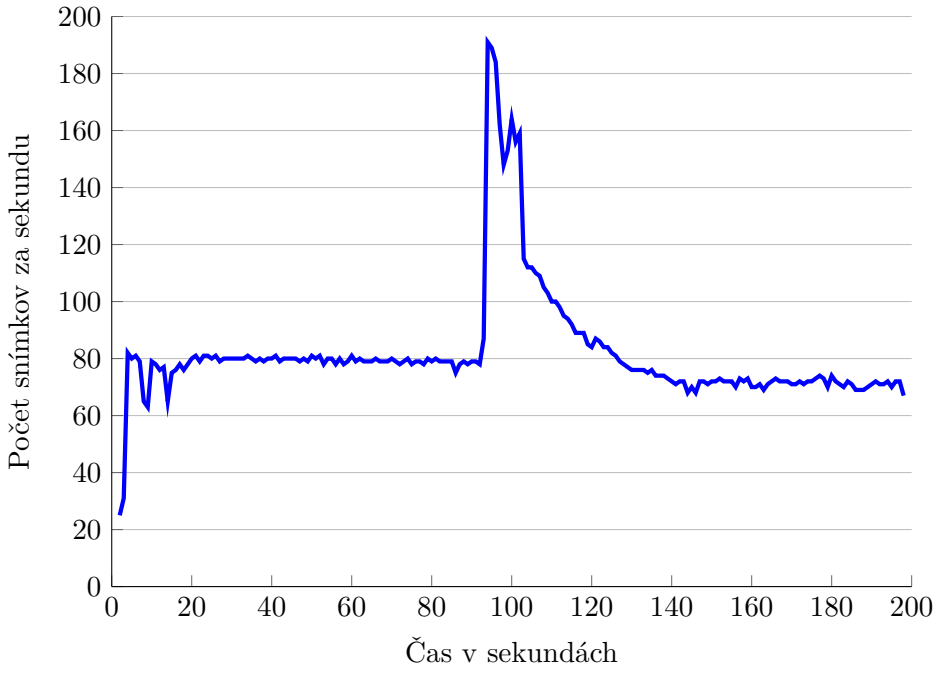
Tabuľka 5.1: Porovnanie výsledkov komprimácií

Všetky komprimačné programy boli schopné redukovať veľkosť výslednej aplikácie na menej než 64 kB. Nakoniec som sa rozhodol použiť UPX 3.94w. Pomocou MPRESS som síce dosiahol lepšie výsledky, ale antivírové programy na viacerých počítačoch takto zkomprimovaný súbor okamžite odstránili.

Kapitola 6

Vyhodnotenie

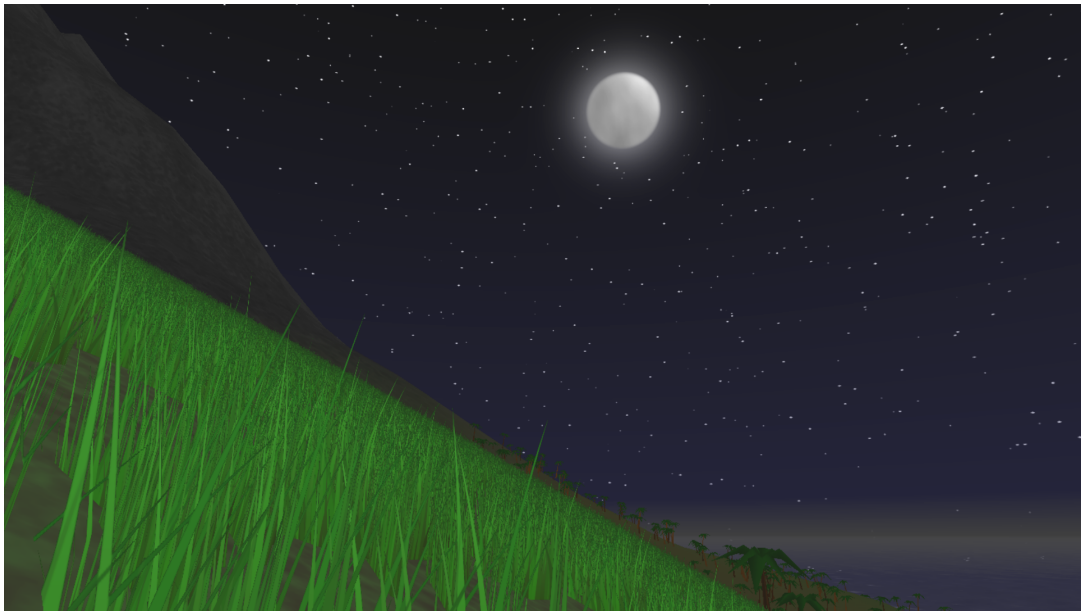
Grafické intro bolo vytvorené a otestované na notebooku DELL Inspiron 7559 s procesorom Intel Core i5-6300HQ a grafickou kartou NVIDIA GeForce GTX 960M. Po spustení sa zobrazí úvodné okno s možnosťami zmeny rozlíšenia. Predvolené rozlíšenie grafického intra je 1024 x 768. V tomto rozlíšení bolo aj grafické intro otestované. Výsledné intro na testovanom počítači dosahovalo hodnôt vyšších ako 60 FPS ako to aj vyplýva z grafu 6.1. Skok nad 180 FPS nastáva v momente keď sa vypne vykresľovanie trávy, ohniska, a zapne sa generovanie častíc lávy. Pri generovaní častíc lávy nám postupne klesá aj FPS. Nakoniec sa však ustáli na podobných hodnotách ako na začiatku intra. Výsledné intro bolo optimalizované na hodnoty 60 FPS a preto sa odporúča nastaviť aj tento limit na 60 Hz monitoroch pomocou vertikálnej synchronizácie.



Obr. 6.1: Graf závislosti počtu snímkov za sekundu na časovom priebehu intra

Vytvorené grafické intro začína pohľadom na ostrov. Po jeho obkrúžení sa priblíži k povrchu ostrova kde je kladený dôraz na palmy a trávu. Odtiaľ putuje k domčeku a ohnisku na demonštráciu časticových systémov. Po nej sa kamera presunie na breh a upriami pohľad

na mesiac. Grafické intro je zakončené pohľadom na stred ostrova, kde začne vybuchovať sopka a vytekať z nej láva. Tu sa čaká na ukončenie programu pomocou tlačidla ESC.



Obr. 6.2: Ukážky z grafického intra

Kapitola 7

Záver

Cieľom tejto bakalárskej práce bolo navrhnúť a implementovať grafické intro, ktoré by nepresiahlo veľkosť 64 kB. Pri tvorbe grafického intra bolo potrebné si naštudovať knižnicu OpenGL ako aj rôzne spôsoby generovania grafického obsahu. Témou na demonštráciu naštudovaných materiálov sa stal sopečný, tropický ostrov.

Na zostrojenie tohoto ostrova boli naštudované a použité metódy procedurálneho generovanie a Perlinov šumu, Phongov osvetľovací model, skybox, časticové systémy, bilboarding, delenie povrchov, zisťovanie kolizií.

Použitím vyššie spomenutých metód sa mi podarilo zostaviť grafické intro s celkovou veľkosťou 58 kB. Čím bola požiadavka na veľkostný limit splnená. Grafická scéna však ani zďaleka nie je dokonalá a dala by sa rozšíriť mnohými spôsobmi. V grafickom intre sa vykresľujú veľké počty objektov, čo na počítačoch zo starším hardvérom môže spôsobiť pokles v počtu snímkov za sekundu. Toto by sa dalo vyriešiť implementovaním frustrum cullingu. Ďalej by scénu bolo možné rozšíriť pridaním hmly, implementáciou shadow-mappingu, implementáciou L-systémov na procedurálne generovanie stromov a rastlín. Keďže tému som si striktne vybral ako prechádzku v noci, nebolo potrebné implementovať animovaný skybox. Ďalším rozšírením by mohlo byť pridanie takéhoto animovaného skyboxu na striedanie dňa a noci.

Literatúra

[1] Arnebäck, E.: ugly grass. [Online; navštívené 10.01.2018].
URL https://www.shadertoy.com/view/MlSXRV

[2] Biagioli, A.: Understanding Perlin Noise. [Online; navštívené 02.12.2017].
URL http://flafla2.github.io/2014/08/09/perlinnoise.html

[3] Bodnar, J.: Windows API tutorial. [Online; navštívené 04.11.2017].
URL http://zetcode.com/gui/winapi/

[4] Creation, G.-T.: OpenGL Mathematics. [Online; navštívené 05.11.2017].
URL http://glm.g-truc.net/0.9.8/index.html

[5] De Vries, J.: Learn OpenGL - Basic Lighting. [Online; navštívené 17.11.2017].
URL https://learnopengl.com/Lighting/Basic-Lighting

[6] De Vries, J.: Learn OpenGL - Instancing. [Online; navštívené 15.02.2018].
URL https://learnopengl.com/Advanced-OpenGL/Instancing

[7] Farbrausch: V2 synthesizer system. [Online; navštívené 11.04.2017].
URL http://1337haxorz.de/products.html

[8] Feldman, B. E.; O'Brien, J. F.; Arikan, O.: Animating Suspended Particle Explosions. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, New York, NY, USA: ACM, 2003, ISBN 1-58113-709-5, s. 708–715.

[9] Horsch, M.: OpenGL Water Tutorial. [Online; navštívené 07.01.2018].
URL http://blog.bonzaisoftware.com/tnp/gl-water-tutorial/

[10] Ikits, M.; Magallon, M.: The OpenGL Extension Wrangler Library. [Online; navštívené 05.11.2017].
URL http://glew.sourceforge.net/

[11] Inc, T. K. G.: Hlavičkový súbor glext.h. [Online; navštívené 21.4.2018].
URL https://www.khronos.org/registry/OpenGL/api/GL/glext.h

[12] Johnson, P.; Urman, M.: The Yasm Modular Assembler Project. [Online; navštívené 11.04.2017].
URL http://yasm.tortall.net/

[13] Jules, C.: Accurate point in triangle test. [Online; navštívené 27.03.2018].
URL http://totologic.blogspot.cz/2014/01/accurate-point-in-triangle-test.html

[14] Meiri, E.: Tutorial 25: SkyBox. [Online; navštívené 07.01.2017].
URL http://ogldev.atspace.co.uk/www/tutorial25/tutorial25.html

[15] Microsoft: Compiler Options Listed by Category. Visual Studio 2015[Online; navštívené 27.04.2018].
URL https://msdn.microsoft.com/en-us/library/19z1t1wy.aspx

[16] Microsoft: Linker Options. Visual Studio 2015[Online; navštívené 27.04.2018].
URL https://msdn.microsoft.com/en-us/library/y0zzbyt4.aspx

[17] Microsoft: Windows API Index. [Online; navštívené 04.11.2017].
URL https:
//msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx

[18] Molofee, J.: Creating an OpenGL Window (Win32). NeHe Productions, [Online; navštívené 04.11.2017].
URL http:
//nehe.gamedev.net/tutorial/creating_an_opengl_window_(win32)/13001/

[19] Reunanen, M.: *Computer Demos—What Makes Them Tick?* Licentiate thesis, AALTO UNIVERSITY, 4 2010.

[20] Schröder, P.; Zorin, D.; Warren, J.; aj.: Subdivision for Modeling and Animation. In *Proceedings of SIGGRAPH 99*, 1999.

[21] Segal, M.; Akeley, K.: The OpenGLR Graphics System: A Specification (Version 4.5 (Core Profile) - June 29, 2017). [Online; navštívené 5.11.2017].
URL https://www.khronos.org/registry/OpenGL/specs/gl/glspec45.core.pdf

[22] Sharp, B.: Subdivision Surface Theory. [Online; navštívené 18.01.2018].
URL http:
//www.gamasutra.com/view/feature/131585/subdivision_surface_theory.php"

[23] Sidelnikov, G.: OpenGL Normal Mapping (DOT3 Bump Mapping). [Online; navštívené 15.01.2018].
URL http://www.falloutsoftware.com/tutorials/gl/normal-map.html

[24] Tagaro, R. F.: ANTI-ALIASING PROBLEM AND MIPMAPPING. [Online; navštívené 10.12.2017].
URL https://textureingraphics.wordpress.com/what-is-texture-mapping/
anti-aliasing-problem-and-mipmapping/

[25] Tan, J. H.; Acharya, U. R.: Active spline model: A shape based model—interactive segmentation. *Digital Signal Processing*, ročník 35, 2014: s. 64 – 74, ISSN 1051-2004.

[26] Teo: downdraft. [Online; navštívené 12.04.2017].
URL https://modland.ziphoid.com/pub/modules/V2/Teo/

[27] Žára, J.; Beneš, B.; Jiří, S.; aj.: *Moderní počítačová grafika*. Computer Press, 2004, ISBN 80-251-0454-0.

Príloha A

DVD

Obsahom priloženého DVD k bakalárskej práci je:

- Zdrojové kódy programu
- Preložený a spustiteľný program
- Zdrojový tvar písomnej správy
- Video bez úpravy
- Video so zvýšeným kontrastom
- Plagát

Príloha B

Plagát